`Content-Type: multipart/oracle`
Tapping into Format Oracles in Email End-to-End Encryption

Fabian Ising
Münster University of Applied Sciences

Damian Poddebniak
Münster University of Applied Sciences

Tobias Kappert
Münster University of Applied Sciences

Christoph Saatjohann
Münster University of Applied Sciences

Sebastian Schinzel
Münster University of Applied Sciences

Abstract

S/MIME and OpenPGP use cryptographic constructions repeatedly shown to be vulnerable to format oracle attacks in protocols like TLS, SSH, or IKE. However, format oracle attacks in the End-to-End Encryption (E2EE) email setting are considered impractical as victims would need to open many attacker-modified emails and communicate the decryption result to the attacker. But is this really the case?

In this paper, we survey how an attacker may remotely learn the decryption state in email E2EE. We analyze the interplay of MIME and IMAP and describe side-channels emerging from network patterns that leak the decryption status in Mail User Agents (MUAs). Concretely, we introduce specific MIME trees that produce decryption-dependent network patterns when opened in a victim's email client.

We survey 19 OpenPGP- and S/MIME-enabled email clients and four cryptographic libraries and uncover a side-channel leaking the decryption status of S/MIME messages in one client. Further, we discuss why the exploitation in the other clients is impractical and show that it is due to missing feature support and implementation quirks. These *unintended* defenses create an unfortunate conflict between usability and security. We present more rigid countermeasures for MUA developers and the standards to prevent exploitation.

1 Introduction

In the last decades, researchers repeatedly presented format oracle attacks such as Bleichenbacher's "Million Message Attack" [6] and Vaudenay's CBC padding oracle attack [44] to break the confidentiality and authenticity of widely used protocols such as TLS [2, 7, 30], SSH [1, 10], and IKE [15]. Even though the two primary standards for email encryption — S/MIME and OpenPGP — use similar cryptographic constructions as TLS, SSH, and IKE, email encryption appears not to be vulnerable to oracle attacks because they require an online oracle that attackers can query. Email allows sending chosen ciphertexts, but its store-and-forward architecture does not allow directly observing the decryption outcome. To learn about the result of the decryption process, the victim would need to cooperate with the attacker, e.g., by manually signaling whether the decryption failed. This process is impractical, especially if many oracle queries are required. Thus, formerly proposed oracle attacks against E2EE email [29, 31] were deemed unrealistic.

```
1  From:          Alice
2  To:            Bob
3  Subject:       Example
4  Content-Type:  multipart/alternative;
5                 boundary=alternative
6
7  --alternative // ---------------------------------------
8    Content-Type: application/encrypted
9
10   [Base64-encoded ciphertext]
11 --alternative // ---------------------------------------
12   Content-Type: application/encrypted
13
14   [Base64-encoded ciphertext]
15 --alternative--
```

Listing 1: A simplified email containing two alternative encrypted MIME parts. A MUA can fetch either part separately via IMAP. It is implementation-specific if fetching a part depends on the decryption result of another part.

Both cases have in common that the OpenPGP community considered *access* to an oracle to be the problem and not the *existence* of the oracle itself, as Maury et al. report in their disclosure results [29]. This attitude led to the fact that OpenPGP and S/MIME did not undergo a rigorous restructuring like TLS 1.3 [38], which prevents many common oracle attacks on the protocol level. The main questions we tackle in the paper are:

Are there remotely accessible side-channels leaking the decryption status in the E2EE email setting? And if not, what prevents them?

1.1 Remote Oracles in E2EE Email

The Efail attacks [35] showed how to exfiltrate plaintext parts under the constraints of the store-and-forward email infrastructure. It used rich-text features of modern Mail User Agents (MUAs) and demonstrated how to obtain plaintext with only a single chosen-ciphertext query to the decryption oracle. Besides this attack, it showed that modern MUAs have functionality that may give feedback to attackers. However, does this functionality offer the required granularity and performance to allow practical Adaptive Chosen-Ciphertext Attacks (CCA2) against E2EE email?

Besides the rich-text features of modern MUAs, other email technologies have the potential for constructing observable oracles. For example, the Multipurpose Internet Mail Extensions (MIME) support the `Content-Type multipart/alternative` to provide multiple *semantically* identical messages in different formats. When an email client cannot load and process a specific MIME part, it may load and try to process another one. If failed decryptions trigger this, attackers may use this for oracle attacks.

To describe this further, let us assume that an attacker provided two encrypted MIME parts (see Listing 1 lines 8–10 and 12–14). The attacker modified the first part to result in a decryption error — for example, because it contains an incorrect PKCS#1 v1.5 padding. If the MUA supports loading alternative MIME parts, it may fetch the second MIME part if and only if the first did not decrypt successfully. These fetches lead to distinct network traffic patterns, depending on the decryption status of the first part. A passive Meddler-in-the-Middle (MitM) can thus send CCA2 messages over SMTP to victims and learn about the decryption status by observing the victims' IMAP network patterns. These patterns are visible through transport encryption like TLS or WiFi encryption. As the MUA performs these actions in the background, the victim may not realize they are under attack.

Additionally, email providers can also implement E2EE email as a server-side feature. As an additional finding, we demonstrate a fully working remote exploit of the CBC padding oracle attack targeting Google's hosted S/MIME in Appendix A. This exploit shows how introducing a single benign-looking change can enable the practicability of CBC padding oracle attacks on S/MIME.

1.2 Attacker Model

E2EE systems must guarantee confidentiality even in the light of a compromise of their transport infrastructure. We assume that the attacker has access to an encrypted victim email and that they send chosen ciphertexts to the victim via SMTP. While the attack can be stealthier if the attacker has access to the victim's IMAP account, this is not required. The ciphertext decryption happens either automatically or manually when the victim opens the email, depending on the MUA and its configuration. The attackers' goal is to decrypt the encrypted email (or parts of it) using an oracle attack.

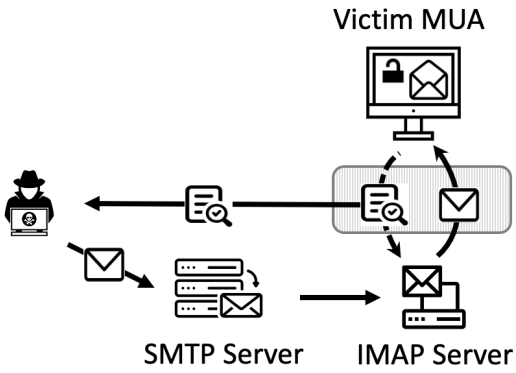

Figure 1: **Attacker Scenario:** The attackers are passive MitM between the victim's MUA and their IMAP server. They can send emails but only observe the (encrypted) traffic between client and server.

We assume that our attackers passively eavesdrop on the encrypted connections between the victim and the IMAP server. They cannot manipulate any traffic between the victim and the mail server. This scenario is visualized in Figure 1.

1.3 Related Work

Security Flaws in email end-to-end encryption. In the last two decades, multiple publications have described attacks against E2EE emails and signatures through different back-channels [33–35]. Poddebniak et al. [35] used ciphertext malleability of the Cipher Feedback (CFB) and Cipher Block Chaining (CBC) encryption modes to produce self-exfiltrating plaintexts. The authors use techniques from MIME and HTML to automatically exfiltrate plaintexts when the user opens the decrypted email.

Attacks published in 2000 [27] and 2019 [34] show that user behavior — i.e., replying to an email — can act as a decryption oracle allowing to attack encrypted and signed emails.

Padding Oracle Attacks. Bleichenbacher's Million Message Attack against SSL was first published in 1998 [6]. Researchers have since adapted the attack to different scenarios [2, 3, 7, 24] not only relevant to SSL/TLS. These attacks include error-, behavior-, and timing-based oracles.

In 2002 Server Vaudenay introduced the CBC padding oracle attack [44], which researchers have since applied to various cryptographic protocols [3, 25, 40].

RFC3218 [37] describes how to mitigate these attacks in the Cryptographic Message Syntax (CMS) — as used by Secure/Multipurpose Internet Mail Extensions (S/MIME). However, both attacks seem particularly hard to avoid, as shown by successful attacks [14, 23] despite countermeasures.

In 2020, Beck et al. [4] presented substantial work on automating the development of Adaptive Chosen-Ciphertext Attacks using format oracles. While they only analyzed symmetric cryptography, it might assist in finding further format oracle attacks in end-to-end encrypted emails.

Format Oracle Attacks on email end-to-end encryption. In 2005 Mister et al. [31] found a format oracle attack against the ad-hoc integrity check functionality — called quick check — in OpenPGP's CFB mode that allows an attacker to determine 16 bits of every plaintext block when accessible.

In 2015, Maury et al. also presented three format oracles against OpenPGP — the Invalid Identifier, the Double Literal, and the MDC Packet Header Oracle [29].

Both the quick check and the other three oracles are only exploitable if the produced error is distinguishable from the integrity check using the Modification Detection Code (MDC).

1.4 Contributions

- We revisit the cryptographic constructions of the S/MIME and OpenPGP standards and analyze the potential for Bleichenbacher and Vaudenay-style format oracles in Section 3. The analysis includes four library implementations used in S/MIME or OpenPGP capable mail clients.
- In Section 3.3, we introduce a new format oracle, the *empty line oracle* found in iOS Mail.
- We survey side-channels arising from the interplay of MIME, IMAP, and SMTP, leaking the decryption status of E2EE email in Section 4.
- The evaluation in Section 5 includes 19 E2EE-capable and widely used MUAs and uncovers several side-channels leaking the decryption status. One side-channel leads to a practical format oracle attack against S/MIME.
- We discuss reasons why most of the tested email clients are not vulnerable to format oracle attacks and why we do not consider these a rigorous defense in Section 6.
- We discuss more rigorous countermeasures for side-channels leaking the decryption status of E2EE email communication in Section 7.
- Server-hosted E2EE email communication may leak the decryption status, leading to online oracles. As an additional finding, Appendix A describes a fully working Vaudenay-style exploit against Google Workspaces.

1.5 Disclosure

We responsibly disclosed all issues, i.e., the Vaudenay Padding oracle in Google Workspaces (see Appendix A) and the empty line oracle in iOS Mail, to the affected vendors.

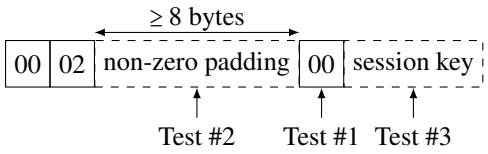

Figure 2: **PKCS#1 v1.5 padding.** Test #1-#3 mark the boolean tests identified by Bardou et al.

Google acknowledged the issue in August 2020 and confirmed that the S/MIME signature check was a countermeasure against Efail. They quickly fixed the problem by not bouncing unsigned messages. Instead, they now mark unsigned emails as suspicious, which disables automatic image loads and serves as a stopgap measure against Efail.

Apple acknowledged the reported issue in October 2021 but, as of September 2022, is still investigating client-side mitigations for a future release.

2 Background

2.1 Oracle Attacks

In an oracle attack, an attacker queries a system with a cryptographic task and observes a *function* of the task's outcome. If the function validates the decrypted plaintext's padding, it is called a padding oracle [3]. More generally, if the function checks the plaintext's format, it is called a format oracle [29].

We define an oracle as the function $O : Q \to A$, where $q \in Q$ is a query an attacker can send to the oracle (which contains one or multiple ciphertexts the attacker wants to decrypt), and $a \in A$ is *some* response they get in return.

In 1998, Daniel Bleichenbacher presented a padding oracle attack, also known as the "Million Message Attack" [6]. This attack targets the PKCS#1 v1.5 padding scheme of RSA-encrypted session keys, as presented in Figure 2. As the name implies, the original attack required up to 2 million oracle queries to decrypt an RSA message with a 1024-bit modulus.

However, Bardou et al. improved Bleichenbacher's attack and reduced the number of queries by a factor of four on average [3]. Also, [3] introduced a framework to assess the usefulness of a PKCS#1 v1.5 padding oracle for an attacker. They define three boolean tests — marked 'T' (test not applied) or 'F' (test applied) — that an implementation can perform on the padding of the session key. Test #1 checks for a zero-byte after the non-zero padding, Test #2 checks if the padding contains any zero-bytes, and Test #3 checks the session key length (see Figure 2). This framework gives an estimation of how many oracle queries are required to decrypt a session key. For example, a "TTT oracle" — no tests applied — means that an attack is possible with 9.374 queries on average, while an "FFF oracle" — all tests applied — means that an attack requires around 2^{26} queries on average

and is less useful for an attacker. Generally, a test that is not applied (T) reduces the number of oracle queries required.

In 2002, Serge Vaudenay presented another famous padding oracle attack against the Cipher Block Chaining (CBC) padding scheme [44]. An attacker can extract the plaintext of an encrypted message merely by using information about the correctness of the padding after message decryption. His oracle takes a ciphertext and returns true if the corresponding plaintext has correct padding. Access to such an oracle allows an attacker to reconstruct the plaintext with an average of 128 queries per plaintext byte [44].

Prerequisites and Exploitability It is often difficult to turn a format oracle into a working exploit. In their seminal work, Beck et al. presented a method to automate the process [4]. For example, the authors constructed a format oracle attack based on the property if a ciphertext decrypts to a valid or invalid Sudoku field. While this seems an academic amusement, it shows an important fact: format oracles may come in different shapes, and it is hard to reason about their exploitability.

Thus, modern encryption technologies exclude this possibility right from the beginning by using authenticated encryption to make a cryptographic scheme non-malleable. Therefore, a decryptor *must* detect a modified ciphertext before releasing plaintext data and reject the message accordingly. This way, an attacker will only learn that any ciphertext $c' \neq c$ is not valid, but they will get no information about the plaintext because no decryption happens in the first place. While this approach may still be implemented incorrectly in practice, the consensus is that a decryptor should release no unauthenticated data before verifying the ciphertext.

The two primary standards for email encryption, S/MIME [36] and OpenPGP [9], do not provide this guarantee, and multiple oracle attacks on S/MIME and OpenPGP were demonstrated in the past [21, 27, 29, 31, 35].

2.2 Email Ecosystem

IMF and MIME In its earliest form, an email is an ASCII-based message conforming to the message format defined in RFC561 [5]. This format initially became an internet standard in [12] and is now referred to as the Internet Message Format (IMF) [39]. Although the IMF was already created in the early '70s, it has not changed substantially. Most notably, it is still a line-based format where an empty line separates the email headers from the body.

Users' expectations towards modern emails changed over time, so the IMF had to include structured and multimedia data. Because the IMF did not describe the body of a message but merely the header format, this could be easily achieved by augmenting IMF with an orthogonal format, the Multipurpose Internet Mail Extensions (MIME) [16–19, 32]. Thus, today, most emails are described by the IMF (defining the header structure) and MIME (defining the body structure).

2.3 IMAP

The Internet Message Access Protocol (IMAP) [11] is a versatile protocol for accessing and organizing messages on a server. IMAP has many useful features, such as server-side search capabilities and the option to fetch only specific parts of a message. Users can use this feature, for example, when accessing the Internet through a metered connection to download only the text of a message and not any attached files. In contrast to its predecessor, POP3, IMAP is also suitable for synchronizing messages on multiple devices because messages are stored permanently on an IMAP server.

2.4 S/MIME and CMS

S/MIME (Secure/MIME) [41] describes sending and receiving cryptographically secured MIME data. It focuses on the "MIME-related parts" of an email, e.g., it defines new MIME media types but relies on the Cryptographic Message Syntax (CMS) [22] as its underlying container format to digitally sign, authenticate, or encrypt arbitrary messages. Corporations and universities commonly use it for end-to-end encrypted email.

2.5 OpenPGP

Privacy advocates and companies commonly use OpenPGP [9] as an alternative to S/MIME. The OpenPGP standard does not define how to encrypt email messages using OpenPGP but, like the CMS, provides a container format that can be used as a general-purpose framework to construct encrypted and signed messages. An additional standard [13] defines embedding OpenPGP messages into the MIME format.

Current OpenPGP implementations usually encapsulate encrypted emails in the "Symmetrically Encrypted Integrity Protected Data packet". For integrity protection, the encrypted payload of this package contains the Modification Detection Code (MDC) packet, which is a SHA1 hash over the plaintext.

3 Format Oracles in Email E2EE

As a prerequisite for our MIME-based format oracle attacks on email end-to-end encryption, we analyze the most common (library) implementations of S/MIME — i.e., Network Security Services (NSS) and GPGME (GPGSM) — and OpenPGP — i.e., GnuPG and OpenPGP.js — for format oracles. We focus on the two most common padding oracles — the CBC padding oracle [44] and the Million Message Attack [6] — as practical examples that nobody has applied to email yet. In addition, we describe a novel format oracle we found in iOS Mail: the empty line oracle.

We summarize the results of the library analysis in Table 1 and link the relevant library code sections in Appendix C.

Application / Library	CBC Padding Oracle	Million Message Attack Oracle	Million Message Attack Query Count
OpenPGP			
GnuPG	–	–	2^{46}
OpenPGP.js	–	–	2^{46}
S/MIME			
NSS	N	FFF	2^{26}
NSS[1]	–	FFT	2^{19}
GPGME	C	FTF	2^{26}
GPGME[1 2]	–	FFT	2^{19}

N CBC padding not checked.
C CBC padding checked.
– Not applicable.

[1] Replacing the algorithm with one with variable key length.
[2] Variable key length algorithms have to be explicitly activated.

Table 1: **Summary of findings for library code.** Oracle strength of the Million Message Attack as defined by Bardou et al. [3]. Query count estimations are worst-case based on the original algorithm. The improved version of the algorithm provides significantly better mean and median counts.

Other Known Format Oracles in OpenPGP We re-evaluated the Quick Check oracle [31] and the three oracles found by Maury et al. [29] against modern email clients. We found that, in practice, no client differentiates between different error cases, leaving us unable to exploit these errors.

3.1 Padding Oracle Attack on CBC Padding

Generally, the CBC padding oracle attack is difficult to prevent if CBC padding is in use and access to a good oracle is available. Therefore, the primary aspect to focus on is the usage and implementation of padding checks in the underlying standards of Secure/Multipurpose Internet Mail Extensions (S/MIME) and OpenPGP.

OpenPGP in current versions [9] implements encryption using the Cipher Feedback (CFB) mode, which requires no padding. Thus, the padding oracle attack on CBC does not apply to OpenPGP.

S/MIME, in the most widely deployed version 3.2, on the other hand, uses encryption in CBC mode [36] without integrity protection. Therefore, S/MIME version 3.2 implementations are potentially vulnerable to the CBC padding oracle attack in the presence of a good oracle — i.e., one that signals correct or incorrect padding. The new S/MIME version 4.0 still requires implementations to support AES-128-CBC, although AES-GCM is recommended [41].

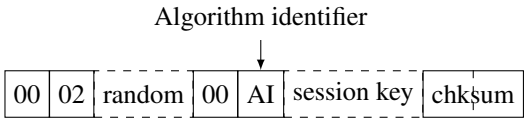

Figure 3: **OpenPGP Session Key:** Content of the RSA encrypted session key.

Limitations The CBC padding oracle attack requires the attacker to adapt queries with the results of previous queries. Since our attacker cannot directly manipulate messages during the IMAP connection between the Mail User Agent (MUA) and the email server, they can only adapt queries with each new email, requiring at least one email per byte of the original message the attacker wants to obtain. An attacker can reduce the number of queries by stacking queries for bytes in different ciphertext blocks since adaptation is only necessary for bytes of the same block.

While this would require at least 256 body parts per guessed byte, we found that no MUA capable of displaying composite messages restricts the number of parts in a single email. Therefore, guessing two bytes of a message, albeit from different ciphertext blocks, would require the attacker to send at least one email.

NSS NSS does not check the full CBC padding of an S/MIME message but only uses the last padding byte, effectively preventing the CBC padding oracle attack.

GPGME (GPGSM) GPGSM thoroughly checks the CBC padding of S/MIME messages.

3.2 Million Message Attack

In the presence of a good oracle, any implementation of PKCS#1 v1.5 is potentially vulnerable to the Million Message Attack [6]. However, the attack's efficiency depends on the format checks performed when verifying the RSA decrypted session key [3].

Notably, the OpenPGP RFC [9] extends the typical usage of PKCS#1 v1.5 by adding additional values to the encoded session key. The plaintext is PKCS#1 v1.5 encoded and starts with a one-byte identifier of the symmetric encryption algorithm, followed by the actual key and a two-byte checksum (see Figure 3). Format checks, i.e., if the algorithm is valid or the checksum is correct, reduce the chance that a "random" byte sequence is correct according to this extended-PKCS#1 v1.5 format, making the oracle less valuable to an attacker.

In contrast, because the S/MIME content-encryption algorithm is not protected, an attacker can change it to a variable key length algorithm, reducing the impact of the length check on the decrypted session key.

Algorithm 1 Simplified algorithm to decrypt a block using the empty line oracle. $block_{-1}$ is the previous ciphertext block, $|x|$ returns the byte length of x, indexing of byte arrays starts at zero, and $||$ is a concatenation of bytes. $oracle(b)$ returns true if the decryption of block b contains an empty line.

```
function DECRYPT_BLOCK(block)
    known_bytes ← DECRYPT_FIRST_BYTES(block)
    for i ← 2 to |block| do
        for g ← 1 to 255 do
            mask ← 0^(i−1) || known_bytes[i − 1] ⊕ 0x10
            mask ← mask || g ⊕ 0x0a || 0^(|block|−i)
            query ← block_{−1} ⊕ mask||block
            if oracle(query) == true then
                known_bytes ← known_bytes || g
                break
            end if
        end for
    end for
    return known_bytes
end function
```

GnuPG GnuPG does not check for the leading zero-byte on the session key because Multi-Precision Integers (MPIs) do not support leading null bytes. It tests for the 0x02 byte but does not check that the random padding is at least 8 bytes long. However, it checks if the encoded value is longer than 8 bytes. GnuPG also tests whether the algorithm identifier, key length, and checksum are valid. Therefore, the oracle on GnuPG would be considerably weaker than an FTF oracle, making the Million Message Attack against GnuPG infeasible.

OpenPGP.js OpenPGP.js' PKCS#1 v1.5 decoder correctly checks for both the leading zero and the 0x02. It also validates the random padding length and the zero-byte separator's presence. OpenPGP.js checks the algorithm identifier, the key size, and the key checksum on the decoded value, resulting in the same oracle strength as GnuPG.

NSS NSS' PKCS#1 v1.5 decoder checks the secret key's prefix for any zero-bytes in the mandatory padding and a zero somewhere after the first ten bytes. The key length check depends on the symmetric algorithm: NSS checks the decrypted session key's size for algorithms with a strict key length requirement, making NSS an FFF oracle. However, if a variable key length algorithm, e.g., RC4, is used, the key length is not verified. Because the S/MIME content-encryption algorithm is not protected, an attacker can change it to a variable key length algorithm and make NSS an FFT oracle.

GPGME (GPGSM) Like GnuPG's OpenPGP implementation, its S/MIME implementation (gpgsm) ignores the leading zero-byte in the PKCS#1 v1.5 padding due to the big int library. GPGSM checks if the block type byte is 0x02 and if the non-zero padding is empty. However, it does not check the minimum padding length. The length of the contained key must be compatible with the content-encryption algorithm. While it is theoretically possible for an attacker to use a content-encryption algorithm with variable key length (e.g., RC4) to circumvent this check, we found that typical distributions of GPGME do not support any variable key length algorithms. Therefore, an oracle based on GPGME will typically be an FTF oracle.

Algorithm 2 Simplified algorithm to decrypt the first two bytes of a block using the empty line oracle.

```
function DECRYPT_FIRST_BYTES(block)
    for i ← 1 to 255 do
        for j ← 1 to 255 do
            mask ← i ⊕ 0x0a || j ⊕ 0x0a || 0^(|block|−2)
            query ← block_{−1} ⊕ mask||block
            if oracle(query) == true then
                return i || j
            end if
        end for
    end for
end function
```

3.3 Empty Line Oracle

The plaintext of a well-formed email has head and body areas separated by an empty line. If an implementation validates this and signals the result to an attacker, it constitutes an exploitable format oracle. This format oracle — as present in iOS Mail — checks for two consecutive line breaks, represented by either two `\r\n` or two `\n`.

An attacker can exploit the empty line oracle, similar to how they would exploit a CBC padding oracle. We present a simplified version of the algorithm that decrypts a single block in Algorithm 1. The attacker performs the following steps, as shown in Algorithm 2, to learn the first two bytes of any ciphertext block. First, the attacker chooses a ciphertext block to attack[1]. Second, the attacker iterates through the first two bytes of the ciphertext by XORing a counter to the previous block. Third, only if the oracle signals successful decryption the attacker knows that the first two bytes are `\n\n`[2]. By XORing the original mask, the attacker now learns the original value of these bytes.

After learning the first two bytes of a block, the attacker can continue the attack, as shown in Algorithm 1. They XOR the second (now known) byte to `\n` and iterate over the third

[1] If the chosen ciphertext block happens to include two consecutive line breaks, the second part of the attack can be performed instantaneously with slight modifications.

[2] Theoretically, the first four bytes may be `\r\n\r\n`, which can easily be checked in a single query by masking the third byte.

```
From:          Alice
To:            Bob
Subject:       Example
Content-Type: multipart/alternative;
              boundary=alternative

--alternative // ---------------------------------------------
  Content-Type: text/plain

  Plain text body.

--alternative // ---------------------------------------------
  Content-Type: text/html

  <b>Fancy HTML body.</b>

--alternative--
```

Listing 2: A basic multipart/alternative message, containing two MIME parts. The indention of the MIME parts is only for readability purposes.

```
C: A FETCH 2 (BODYSTRUCTURE)
S: * 2 FETCH (
  BODYSTRUCTURE
  (
    // The first part is a text/plain email.
    ("TEXT" "PLAIN" NIL NIL NIL "6BIT" 20 2)
    // The second part is a text/html email.
    ("TEXT" "HTML" NIL NIL NIL "6BIT" 27 2)
    // Parts are composed via multipart/alternative.
    "ALTERNATIVE" ("BOUNDARY" "alternative") NIL NIL
  )
)
S: A OK fetch done.
C: B FETCH 2 (BODY[2])
S: * 2 FETCH (BODY[2] {25}
   <b>Fancy HTML body.</b>
   )
S: B OK fetch done.
```

Listing 3: Example IMAP flow of a MUA fetching a part of the multipart/alternative email from Listing 2.

until they hit `\n\n`. By XORing the third byte of the mask with `\n`, they learn the third byte. The attacker can repeat this process until an entire plaintext block is known.

Decrypting a 16-byte ciphertext block using this format oracle requires a substantial number of queries (34,560 on average or 9,088 under the assumption that the email only contains ASCII characters) when performing a naive sequential search. The attacker must execute the attack in its entirety for each ciphertext block. However, this oracle attack is practical to decrypt short emails or emails with short blocks of interest (e.g., password reset codes).

As an optimization, an attacker can batch multiple queries in a single email to perform a binary search, as exemplified in Figure 4, reducing the search space dramatically. However, this optimization causes false positive oracle results due to the random blocks introduced by manipulating a CBC ciphertext. While these false positives have to be re-validated with some backtracking, the optimized attack is still dramatically (about factor 20 for a 16-byte ASCII block) faster.

While this oracle is independent of the used encryption

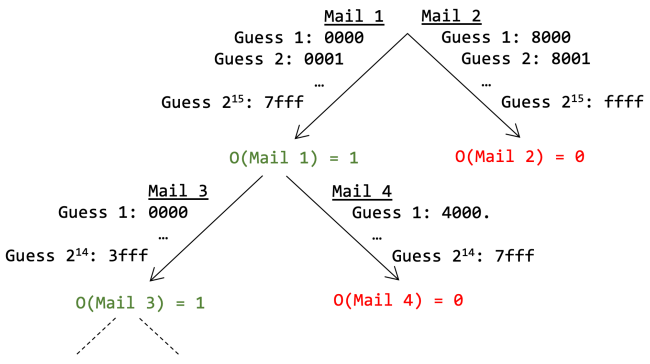

Figure 4: **Empty Line Oracle:** Batching multiple oracle queries to perform a binary search of the first two bytes of a block. The guesses of each mail are in a single message part.

mechanism — as it is MUA specific — the attack is prevented in OpenPGP if the implementation checks the MDC.

4 MIME-Based Oracles

Recent attacks [35] show that the content and context of encrypted emails can strongly affect the confidentiality of the message when rendered by MUAs. Especially the Efail direct exfiltration attacks show that unsafe rendering of MIME emails enables severe attacks on email encryption. Therefore, we analyze the relevant MIME standards in search of potential oracles, i.e., via external requests or IMAP commands.

4.1 General Interaction of MIME and IMAP

We found that MUAs change how they download and display emails depending on the MIME message structure. While this is not surprising when considering the display of external content, email structure can even influence the IMAP message flow between the email server and the MUA.

While a MUA can download complete messages without taking structure into account, the IMAP protocol allows fetching specific message parts, e.g., to download only the plain text content from a `multipart/alternative` email (see Listing 2 and Listing 3).

Our IMAP standard analysis shows two instances of interaction with the MIME structure. Both allow downloading a MIME message with separate `FETCH` commands.

First, a MUA can request the message structure via IMAP's `BODY` or `BODYSTRUCTURE` fetch attribute. Second, the MUA retrieves the actual content of a MIME message, which it can do via partial fetching of specific message parts (Listing 3).

Some MUAs use partial fetching for all composite messages, not only `multipart/alternative` emails. For other messages, parts of the message are fetched consecutively instead of in bulk. We call this behavior *lazy fetching* in contrast to fetching complete emails at once — *greedy fetching*.

```
Content-Type: multipart/alternative;
              boundary=alternative

--alternative // ----------------------------------------------
  Content-Type: text/html

  <img src=attacker.example.org/ping>

--alternative // ----------------------------------------------
  Content-Type: application/pkcs7-mime;
                smime-type=enveloped-data

  [Base64-encoded S/MIME encrypted message]

--alternative--
```

Listing 4: Multipart/alternative email containing both an S/MIME encrypted and an HTML body part. From now on, we will only provide the headers relevant to our work.

Additionally, we distinguish between two methods of displaying content. A MUA can process all body parts of a message in bulk and display them afterward — *greedy rendering* — or choose to render parts of an email as soon as the processing (e.g., decryption) is complete — *lazy rendering*.

4.2 Composite Messages

Since meaningful interaction regarding message structure requires emails more complex than a single message body, we analyzed several MIME composite message formats [18] for potentially useful behavior. Since the discrete media types — e.g., `text/plain` — "must be handled by non-MIME mechanisms [and] are opaque to the MIME processors" [18], we assume that these are less relevant when looking at the behavior of MIME processors — i.e., MUAs. However, the MUA typically handles the composite media types directly. Therefore, its behavior can change depending on the structure.

MUAs mainly use the top-level media type `message` to encapsulate email messages inside an email. For example, they often use `message/rfc822` messages for forwarding emails. `Message/partial`'s primary use is to split large messages, e.g., in cases where an intermediate Mail Transfer Agent (MTA) restricts message sizes. Since all these messages represent full MIME messages that a MUA can only request in full, we assume they are handled as single entities and provide no further information to an attacker.

However, the `multipart` media types might lead to observable network patterns. The security subtypes `signed` and `encrypted` [20] enable cryptographic operations, whereas the other subtypes allow for complex message construction.

Multipart/alternative The `alternative` subtype allows bundling multiple *alternative* representations of the same message — usually using different Content-Types — in case the receiver cannot display a specific message type. A standard-compliant MIME parser should first try to display the last part continuing in ascending order until they find a representation they can show. However, combined with encrypted message

```
Content-Type: multipart/mixed;
              boundary=mixed

--mixed // ----------------------------------------------------
  Content-Type: text/html

  Unencrypted message part 1.
  <img src=attacker.example.org/ping>

--mixed // ----------------------------------------------------
  Content-Type: application/pkcs7-mime;
                smime-type=enveloped-data

  [Base64-encoded S/MIME encrypted message]

--mixed // ----------------------------------------------------
  Content-Type: text/html

  Unencrypted message part 3.
  <img src=attacker.example.org/pong>

--mixed--
```

Listing 5: Multipart/mixed email containing an S/MIME encrypted and two text/html body parts.

parts, this process potentially leads to observable behavior from MUAs.

Presented with the message shown in Listing 4, a MUA capable of decrypting S/MIME messages would start by processing the second body part. If processing this body part leads to an error — i.e., a failed format check — the MUA might show the `text/html` body instead. If the MUA implements lazy fetching, an attacker can observe the retrieval of this message part, indicating that the MUA could not decode the encrypted body part, leading to the following oracle:

$$O(c) = \begin{cases} \text{decryption failed} & \texttt{FETCH BODY[1]} \\ \text{decryption succeeded} & \text{otherwise.} \end{cases}$$

An attacker can also observe this behavior if they are *not* a MitM between the IMAP server and the MUA, if the MUA loads external content. In this case, the MUA will only request an image loaded in the first body part if they cannot decrypt the second part. This leads to the following oracle (where the server at attacker.example.org observes the `GET` request):

$$O(c) = \begin{cases} \text{decryption failed} & \texttt{GET ping} \\ \text{decryption succeeded} & \text{otherwise.} \end{cases}$$

This oracle is even present if the MUA does not employ lazy fetching but only employs lazy rendering.

Multipart/mixed With the `mixed` subtype, a sender can bundle independent message parts in a particular order in a single email. Since the sequence of message parts is strictly defined, we assume that any standard-compliant MIME parser processes these parts in the given order. However, this leads to interesting behavior when dealing with errors in cryptographic operations — i.e., a failed format check.

Consider a message as shown in Listing 5. If the encrypted body part decrypts without errors, the MUA will display all

```
Content-Type: multipart/related;
              boundary=related

--related // ------------------------------------------------
  Content-Type: text/html

  // Include other parts in iFrames using Content-ID (cid).
  <iframe src=cid:ping>
  <iframe src=cid:encrypted>
  <iframe src=cid:pong>

--related // ------------------------------------------------
  Content-Type: text/html
  Content-ID:   <ping>

  <img src=attacker.example.org/ping>

--related // ------------------------------------------------
  Content-Type: text/html
  Content-ID:   <pong>

  <img src=attacker.example.org/pong>

--related // ------------------------------------------------
  Content-Type: application/pkcs7-mime;
                smime-type=enveloped-data
  Content-ID:   <encrypted>

  [base64-encoded S/MIME encrypted message]

--related--
```

Listing 6: A multipart/related email with an encrypted part and two unencrypted parts with external content.

parts in the given order. However, if the decryption of the encrypted body part fails and lazy fetching is in use, several potential timing side-channels emerge.

Since the attacker can observe the lazy fetching behavior, they can measure the time between the MUA's `FETCH` request for the encrypted body part and the request for the HTML body part. Depending on the decryption process, this time can differ if a format check fails — e.g., the MUA performs no symmetric decryption due to a failed format check on the asymmetrically encrypted session key. Even if there is no measurable time difference in the cryptographic operations due to a failed padding check, the time necessary to render a correctly decrypted ciphertext can be observable. These measurements result in the following oracle, assuming a sufficient threshold t_{suc} for successful decryptions has been determined:

$$O(c) = \begin{cases} \text{decryption failed} & \Delta t < t_{suc} \\ \text{decryption succeeded} & \text{otherwise.} \end{cases}$$

where

$$\Delta t = \text{time}(\texttt{FETCH BODY[3]}) - \text{time}(\texttt{FETCH BODY[2]}).$$

Notably, this oracle can also be observed by a non-MitM attacker if the MUA employs lazy rendering and displays external content by checking the timing between the `GET` requests using the same oracle with

$$\Delta t = \text{time}(\texttt{GET pong}) - \text{time}(\texttt{GET ping}).$$

Multipart/related `related` messages bundle message parts that have some internal linkage between parts. Since these links are in a defined order, an attacker can potentially use this content type to link encrypted parts with unencrypted external content that leaks timing or error information to the attacker. We present an example of this in Listing 6. The oracle is the same timing-based oracle as the `mixed` oracle.

Additionally, an attacker could mount a more complex attack using crypto gadgets [35], transforming the symmetrically encrypted message into a similar `multipart/related message` and observing if related parts are processed. However, since crypto gadgets would also allow the Efail exfiltration attacks, we exclude this approach from our analysis.

Multipart/parallel The `parallel` subtype is often handled the same as `mixed`. However, it allows a MUA to load and display all parts in parallel instead of serial processing. Since parallel display and decryption would make timing measurements more complex and add more jitter, it is unsuitable for timing-based attacks. Also, since error conditions in one part should not affect the processing of other parts, we did not analyze the `parallel` subtype in detail.

Multipart/digest MUAs can use `Multipart/digest` messages to combine multiple parts of type `message/rfc822` into a single message, e.g., to create mailing list digests. Since, according to RFC 1341 [8], `multipart/digest` messages should, in other regards, be handled the same as `multipart/mixed`, we excluded it from further analysis.

4.3 Optimizations

Especially when using timing measurements as oracles, the actual duration of an operation is a critical factor in increasing the measured operation's Signal-To-Noise-Ratio. For example, the time a MUA requires to decrypt a message symmetrically correlates with the size of the given encrypted message. By padding the (not integrity-protected) ciphertext with arbitrary data, an attacker can increase the required decryption duration of the MUA in relation to the network jitter.

Even if no specific attacks using composite message types are possible, they can still improve upon single-part attacks. An attacker can batch multiple oracle queries using the `multipart/mixed` content type, reducing the effective number of emails necessary for a successful attack.

5 Client Evaluation

To evaluate where oracle attacks against email E2EE are feasible, we performed a structured analysis of 19 real-world email clients against these attacks.

Client	Multiple Encrypted Parts	Fetching Behavior: Body (Parts)	Fetching Behavior: Lazy	Decryption	Practical Exploit
Required for practical exploit	✓	●	✓	●	
Clients not supporting multiple encrypted parts					
Airmail	–	●	–	◑	□
eM Client	–	◑	✓	?	□
Mail (macOS)	–	●	~	●	□
MailDroid	–	●	✓	○	□
Nine	–	●	–	◑	□
Outlook 2016	–	●	–	?	□
Outlook 2019	–	●	–	?	□
Postbox	–	●	–	?	□
R2Mail2	–	◑	✓	◑	□
Thunderbird	–	●	–	?	□
Clients not automatically fetching single body parts					
Claws	✓	◑	–	◑	□
Horde IMP	✓	◑	✓	◑	□
Evolution	✓	◑	–	◑	□
KMail	✓	◑	–	○	□
Mutt	✓	◑	–	○	□
The Bat!	✓	◑	–	○	□
Trojitá	✓	◑	✓	◑	□
Clients not using lazy fetching					
MailMate	✓	●	–	◑	□
Clients fulfilling all criteria					
Mail (iOS)	✓	●	✓	●	■

Legend					
✓	Yes	●	Automatic in background.	■	Found
–	No	○	Needs explicit user interaction.	□	Not found
~	Situation dependent	◑	Upon opening email.		
		?	Not detectable.		

Table 2: **Results of our evaluation of email clients**. We report if clients support encrypted mails with multiple parts, if they employ lazy fetching, and when they decrypt messages. The final column indicates if we found a practically exploitable oracle. Greyed out results are included for completeness, but are not directly relevant for exploitability.

5.1 Client Selection

As shown above, even in the presence of an oracle, the Million Message Attack is impractical against common OpenPGP implementations, and no CBC padding is employed. Therefore, we restrict our analysis to email clients supporting S/MIME encryption, which is potentially vulnerable to both attacks.

We selected clients that support S/MIME based on prior work in [35] and excluded long outdated clients. We present details on the tested clients in Table 3 in Appendix B. We performed all tests in the clients' default configuration.

5.2 Criteria for Successful Attacks

We evaluated the selected clients based on three criteria: support for multipart messages with encrypted parts, fetching behavior, and decryption behavior. We filtered the clients step by step according to these criteria until one client — iOS Mail — remains that fulfills all of them, and we will present it as a case study. The detailed results are summarized in Table 2.

A Note on External Content: In addition to our previously described attacker scenario, we evaluated a weaker attacker scenario where the attacker is not a MitM but sends emails containing external content to the victim. We found that many clients do not load external content without user interaction

for privacy reasons, limiting the usefulness of this attacker scenario. Therefore, we exclude this scenario from further evaluation. We present results on this in Appendix B.

Multiple Encrypted Parts The first requirement for practical oracle attacks against MUAs is the support of emails containing multiple encrypted parts. Ten of the tested clients did not support multiple encrypted message parts. In most cases, MUAs displayed additional parts as attachments or did not display the message at all. This requirement left us with nine clients to focus on, as shown in the Multiple Encrypted Parts column in Table 2.

Fetching Behavior The described oracles require specific behavior on the IMAP channel. First, clients should fetch email contents as soon as they are available on the IMAP server to allow for automatic oracles. Additionally, MUAs must use lazy fetching to employ the described techniques based on multipart messages.

We constructed multiple test cases to determine MUAs' fetching behaviors. The tests consist of `multipart` emails with an increasing number of parts (up to 100) and part sizes (up to 10MB total mail size). We summarize the results of these tests in the Fetching Behavior columns in Table 2.

We found that the client behavior is almost evenly split between downloading the message in the background (10 clients) and fetching the email body when the user opens it[3] (9 clients). Most desktop MUAs use greedy fetching. Except for eM Client, Trojitá, and macOS's Mail, all desktop mail clients only fetch complete messages from the IMAP server. MacOS Mail switches from greedy fetching to lazy fetching when the message contains at least 20 message parts and at the same time is larger than 5 MB.

On the other hand, mobile clients preferred lazy fetching, presumably due to possibly flaky mobile data connections. The same is true for Horde, the only web client tested.

This analysis step already left us with only one client to focus on — iOS Mail.

Decryption Efficient automatic exploitation requires the client to decrypt encrypted parts in between fetching them. In some cases, it is evident when the MUA performs the decryption, i.e., when the user needs to click a decrypt button (e.g., The Bat!), or a decrypted preview is shown, in other cases, we could not detect the decryption behavior.

However, we could determine the decryption behavior of all clients employing lazy fetching. Only iOS and macOS Mail perform instant decryption of mail parts. However, since only iOS Mail supports multiple encrypted parts in a single email, it remains the sole client to analyze.

[3] Usually, they fetch header information immediately to display metadata.

5.3 Case Study: iOS Mail

We perform additional tests to determine if a format oracle is exploitable in iOS Mail by crafting emails with multiple parts and observing the fetching behavior. Specifically, we crafted emails containing multiple unmodified ciphertexts, emails containing only manipulated ciphertexts, and emails containing both.

iOS Mail shows easily distinguishable behavior on the IMAP channel for failed format checks. It employs lazy fetching to download `multipart/mixed` emails and stops fetching other body parts (with some delay) if a part fails to decrypt. An attacker can reliably observe this using emails with 100 identical body parts. The resulting oracle is of the form

$$O(c) = \begin{cases} \text{decryption succeeded} & \text{all body parts fetched} \\ \text{decryption failed} & \text{otherwise.} \end{cases}$$

This oracle is limited to one query per email. However, since Mail fetches and decrypts messages in the background, it can be automated reasonably well.

Despite the presence of this oracle, defect CBC padding cannot be detected since Mail displays corrupted messages for manipulated padding, not triggering the oracle. The Million Message Attack is potentially exploitable, but according to our oracle strength tests, it is an FFF oracle. Therefore, both oracles remain only potentially exploitable.

However, Mail is vulnerable to the empty line format oracle from the previous chapter. While this oracle requires a lot of queries, an attacker can automatically exploit it in the background. This attack is feasible for emails where only short blocks (e.g., reset or two-factor codes) are of interest.

Empty Line Oracle The attack algorithm is the one shown in Algorithm 1. An attacker queries the oracle by sending an email with the same ciphertext (the actual oracle query) duplicated as 100 parts of a multipart mixed message. The attacker then observes the IMAP traffic. If iOS Mail fetched all body parts, the decryption was successful, meaning that the message contained an empty line.

We simulated this attack using an iPhone 13 running iOS 15.6 and a customized email server in a lab setting. The iPhone was idle, the display turned on, and the Mail app was running. First, we performed a naive sequential search to decrypt a single 16-byte block of an email known to contain only hexadecimal characters, meaning 16 possible byte values per plaintext byte. Therefore, the worst-case scenario requires 480 queries for a single block. In 20 runs of the described experiment, it took 11 minutes on average to decrypt a single block. We extrapolate from our measurements that, on average, decrypting a 16-byte ASCII plaintext block takes around 4 hours with this approach.

However, batching multiple possible bytes in a single message part to query the oracle significantly improves the performance of the attack. With batching, a binary search of the

search space, on average, takes only 12 minutes — including necessary backtracking — and at least 224 query emails to decrypt a 16-byte ASCII block.

Interestingly, we found that iOS Mail slows down fetching after downloading 100 to 150 emails, allowing a query once every two seconds. The query rate accelerates if the user uses the device — not necessarily the Mail app. We assume this is for power-saving reasons and noticed it occurs in unpredictable patterns. Our measurements take these slowdowns into account. In practice, an attacker could spread the process over multiple days or sessions to decrypt multiple blocks of an email.

6 Discussion

Our evaluation shows that most MUAs happen to be not vulnerable to practical oracle attacks. However, this is not because of conscious efforts to prevent these attacks. It merely stems from limited support of features and implementation quirks — an observation similar to that of Schneier et al. [26].

Following, we discuss why most MUAs are resistant to format oracle attacks and why we do not consider this resistance to be a rigorous defense.

6.1 Incomplete Implementations in MUAs

Email clients resist oracle attacks mainly because of limited support for specific features. For example, over half of the tested clients did not support multiple encrypted message parts in a single email. While this prevents practical oracle attacks, this hardly seems a conscious choice to mitigate this type of attack.

However, for some clients, it is plausible that this might have been a conscious decision to thwart existing attacks on E2EE emails, such as [33–35].

Another feature of the IMAP standard is the usage of lazy and selective fetching. This feature can drastically improve bandwidth usage, i.e., by not downloading message parts with MIME types that the client cannot display, and usability on slower networks — e.g., not requiring attachments to be downloaded before the user requests them. Unsurprisingly, mainly mobile clients use lazy and selective fetching, as they commonly experience more unstable network environments.

Relying on missing feature support for security is particularly dangerous since developers might implement these features later without considering the ramifications for security. If, for example, a widely used client starts to encrypt the message body and attachments separately, this might force other clients to implement support, too, potentially enabling the presented attacks.

Restriction of Background Behavior Even those MUA that support multiple encrypted message parts and use lazy fetching are not necessarily vulnerable if they do not perform fetching and decryption in the background.

Our analysis shows that only two tested clients verifiably decrypt messages before the user opens them. While this seems a rational choice that drastically reduces the practicality of format oracle attacks, it comes with disadvantages to user experience — such as delayed message display and reduced notification contents.

Furthermore, researchers may discover format oracles in email E2EE that require only a few queries to exploit. Attackers could still exploit these in clients that only fetch or decrypt emails after the user opens them.

6.2 Implementation Quirks

The last interesting accidental defense against format oracle attacks lies in the implementation details of email clients. For example, Mail on iOS is not vulnerable to the Vaudenay Padding oracle attack simply because it does not validate the PKCS#7 padding. It only checks the last byte, causing malformed messages to be displayed. While this prevents the format oracle attack, it can hardly be considered a rigorous defense since it allows for other manipulations — i.e., truncation of the plaintext.

7 Countermeasures

Most clients were not vulnerable to attacks; however, this resistance was hardly due to a conscious choice. Following, we describe the most practical countermeasures that actors involved in the email E2EE environment should take to prevent oracles in the future.

7.1 General Considerations

The most basic way of preventing oracle attacks on any protocol is not to leak the decryption status to the attacker. In practice, this is challenging, and even implementations that care to mitigate specific oracle attacks can still provide subtle side-channels in unexpected circumstances [14].

An attacker must be unable to distinguish the decryption results to prevent format oracles reliably. Indistinguishability includes requiring constant-time operations on all operations related to format checks. As we show in this paper, this even includes seemingly benign factors, such as network operations that only appear for specific decryption statuses. This is particularly challenging for format oracles in asymmetric encryption like Bleichenbacher's Million Message Attack.

For format oracle attacks against symmetric encryption, Authenticated Encryption — ideally Authenticated Encryption with Associated Data (AEAD) — should be used to prevent ciphertext manipulation in the first place.

7.2 Stopgap Fixes in Email Clients

As discussed, incomplete implementations in many MUAs did prevent exploitable format oracles despite the presence of these oracles in OpenPGP and S/MIME. While we think that this creates an unfortunate conflict between usability and security, there is not much else clients can do to prevent oracles until robust fixes are in the standards. Therefore, we present some reasonable feature restrictions that developers could implement as a *conscious* choice for security.

First, for now, it is reasonable not to support multiple encrypted messages inside a single email as, to our knowledge, no MUA sends messages with multiple encrypted parts. This restriction dramatically reduces the attacks' effectiveness by limiting the interaction between the IMAP protocol and encrypted messages. Unfortunately, this prevents users from encrypting and downloading attachments separately.

Second, depending on the user base of the MUA, it might be reasonable to delay decrypting emails until the user opens them rather than automatic decryption in the background. This will prevent oracles that require no user interaction. However, this countermeasure worsens the user experience of email E2EE, for example, by not showing previews of new emails and increasing the time it takes to display encrypted emails.

7.3 Cryptographic Libraries and Standards

Since Bleichenbacher published the Million Message Attack in 1998, researchers have proposed several effective countermeasures that cryptographic library developers should implement. As [7] already highlighted for TLS implementations, for RSA with the PKCS#1 v1.5 padding scheme, the decryption of incorrectly formatted messages must be indistinguishable from correctly formatted messages. For libraries used in email encryption, this mainly includes indistinguishable timing for well- and ill-formed plaintexts.

The most straightforward fixes for format oracles on symmetrically encrypted ciphertexts are Encrypt-then-MAC schemes or authenticated encryption that prevent an attacker from manipulating the ciphertext. Integrity protection mitigates, among others, the CBC padding oracle attack, but not attacks on asymmetric ciphers, such as the Million Message Attack. Switching to a more resilient padding scheme like RSA-OAEP or moving away from RSA is advisable.

For S/MIME, the current 4.0 standard [41] contains helpful security considerations that help mitigate oracle attacks. Among others, the authors recommend the usage of AEAD and treating MIME parts as separate entities. They also at least *recommend* implementing RSA-OAEP and ECDH. Even though these recommendations are reasonable, our research highlights that they should be made even stronger, potentially even enforcing the usage of AEAD ciphers.

Furthermore, the Million Message Attack becomes harder or even impractical to exploit with stricter format checks, as shown in Table 1. Therefore, thorough padding checks without any (performance-driven) shortcuts are critical here.

7.4 MIME-Layer

The presented oracles are only possible because neither OpenPGP nor S/MIME protects the original MIME structure of an E2EE email. The countermeasures proposed by Schwenk et al. [42] add this protection. They suggest the usage of "decryption contexts", a canonicalized string representation of the MIME structure of an email.

With the decryption context as Associated Data in an AEAD scheme, the decryption would break if the MIME structure changed, preventing the presented oracle attacks and allowing to encrypt attachments and text separately securely. Unfortunately, no standard has implemented this so far.

8 Conclusion

Previous work on email E2EE has proven the existence of format oracles and has shown that their accessibility leads to full decryption of plaintexts. This paper focuses on the *accessibility* of format oracles in real-world scenarios. It shows that more elaborate implementations of IMAP and MIME make oracles accessible in E2EE email and allow practical attacks against S/MIME and OpenPGP.

While limited support for IMAP and MIME features in email clients prevents most attacks in one way or another, incomplete implementations are at odds with usability, which creates a conflict between usability and security. Thus, anticipating the implementation of additional features, we argue that actors should consider proactive countermeasures — which we could not observe during our research.

While countermeasures to prevent oracle feedback may improve the security of E2EE email in the short term, they only obscure the existence of oracles. Instead, the malleability of ciphertexts should be considered the root cause of effective oracle attacks and mitigated.

Our work supports the criticism raised by related work that currently deployed E2EE email standards are cryptographically fragile and reinforces the need for better cryptographic primitives in S/MIME and OpenPGP.

Acknowledgments We thank the USENIX reviewers for their insightful comments on this paper. We also thank Uwe Sommer of NetCon Consulting for supporting us in testing Google's hosted S/MIME solution. Fabian Ising was supported by a graduate scholarship from Münster University of Applied Sciences and the research project "SEAN", part of the postgraduate research training group North Rhine-Westphalian Experts on Research in Digitalization (NERD.NRW), funded by the Ministry of Culture and Science of North Rhine Westphalia (MKW NRW). Christoph Saatjohann was supported by the research project "MedMax", part of NERD.NRW, funded by the MKW NRW.

References

[1] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext Recovery Attacks Against SSH. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 16–26, Washington, DC, USA, 2009. IEEE Computer Society.

[2] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 689–706, Austin, TX, August 2016. USENIX Association.

[3] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. Efficient Padding Oracle Attacks on Cryptographic Hardware. In *Advances in Cryptology–CRYPTO 2012*, pages 608–625. Springer, 2012.

[4] Gabrielle Beck, Maximilian Zinkus, and Matthew Green. Using SMT Solvers to Automate Chosen Ciphertext Attacks. Cryptology ePrint Archive, Report 2019/958, 2019. https://eprint.iacr.org/2019/958.

[5] A.K. Bhushan, K.T. Pogran, R.S. Tomlinson, and J.E. White. Standardizing Network Mail Headers, September 1973. RFC0561.

[6] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer, 1998.

[7] Hanno Böck, Juraj Somorovsky, and Craig Young. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 817–849, Baltimore, MD, August 2018. USENIX Association.

[8] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies, June 1992. RFC1341.

[9] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format, November 2007. RFC4880.

[10] Core Security. SSH Protocol 1.5 Session Key Recovery Vulnerability. https://www.coresecurity.com/core-labs/advisories/ssh-protocol-15-session-key-recovery-vulnerability, 2001.

[11] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1, March 2003. RFC3501.

[12] D. Crocker. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC 822, August 1982.

[13] M. Elkins, D. Del Torto, R. Levien, and T. Roessler. MIME Security with OpenPGP, August 2001. RFC3156.

[14] N. J. Al Fardan and K. G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, May 2013.

[15] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The Dangers of Key Reuse: Practical Attacks on IPsec IKE. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 567–583, Baltimore, MD, August 2018. USENIX Association.

[16] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples, November 1996. RFC2049.

[17] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996. RFC2045.

[18] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996. RFC2046.

[19] N. Freed, J. Klensin, and J. Postel. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, November 1996. RFC2048.

[20] J. Galvin, S. Murphy, S. Crocker, and N. Freed. Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted, October 1995. RFC1847.

[21] Matthew Green. What's the matter with PGP?, August 2014. https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/.

[22] R. Housley. Cryptographic Message Syntax (CMS), September 2009. RFC5652.

[23] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Lucky 13 Strikes Back. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 85–96, New York, NY, USA, 2015. ACM.

[24] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, pages 752–769, 2012.

[25] Tibor Jager and Juraj Somorovsky. How To Break XML Encryption. In *The 18th ACM Conference on Computer and Communications Security (CCS)*, October 2011.

[26] Kahil Jallad, Jonathan Katz, and Bruce Schneier. Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG. In Agnes Hui Chan and Virgil Gligor, editors, *Information Security*, pages 90–101, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[27] Jonathan Katz and Bruce Schneier. A Chosen Ciphertext Attack Against Several E-Mail Encryption Protocols. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 18–18, Berkeley, CA, USA, 2000. USENIX Association.

[28] J. Klensin. Simple Mail Transfer Protocol, October 2008. RFC5321.

[29] Florian Maury, Jean-René Reinhard, Olivier Levillain, and Henri Gilbert. Format Oracles on OpenPGP. In Kaisa Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 220–236, Cham, 2015. Springer International Publishing.

[30] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *USENIX Security Symposium*, pages 733–748, 2014.

[31] Serge Mister and Robert Zuccherato. An Attack on CFB Mode Encryption As Used By OpenPGP. Cryptology ePrint Archive, Report 2005/033, 2005. https://eprint.iacr.org/2005/033.

[32] K. Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, November 1996. RFC2047.

[33] Jens Müller, Marcus Brinkmann, Damian Poddebniak, Hanno Böck, Sebastian Schinzel, Juraj Somorovsky, and Jörg Schwenk. "Johnny, you are fired!" – Spoofing OpenPGP and S/MIME Signatures in Emails. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1011–1028, Santa Clara, CA, August 2019. USENIX Association.

[34] Jens Müller, Marcus Brinkmann, Damian Poddebniak, Sebastian Schinzel, and Jörg Schwenk. Re: What's Up Johnny? In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 24–42, Cham, 2019. Springer International Publishing.

[35] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.

[36] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, January 2010. RFC5751.

[37] E. Rescorla. Preventing the Million Message Attack on Cryptographic Message Syntax, January 2002. RFC3218.

[38] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, 2018. RFC8446.

[39] P. Resnick. Internet Message Format, October 2008. RFC5322.

[40] Juliano Rizzo and Thai Duong. Practical Padding Oracle Attacks. In *Proceedings of the 4th USENIX Conference on Offensive Technologies*, WOOT'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[41] J. Schaad, August Cellar, B. Ramdell, and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification, 1998. RFC8551.

[42] Jörg Schwenk, Marcus Brinkmann, Damian Poddebniak, Jens Müller, Juraj Somorovsky, and Sebastian Schinzel. Mitigation of Attacks on Email End-to-End Encryption. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1647–1664, New York, NY, USA, 2020. Association for Computing Machinery.

[43] U.S. Food & Drug Administration. CFR - Code of Federal Regulations Title 21. https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=11&showFR=1, April 2020.

[44] Serge Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 534–546, London, UK, UK, 2002. Springer-Verlag.

```
S: 554 5.7.5 To prevent known S/MIME vulnerabilities,
       Gmail does not accept S/MIME encrypted messages
       without an accompanying valid S/MIME signature.
```

Listing 7: SMTP reply for encrypted messages without a valid signature.

A Attacking Google's Hosted S/MIME

Business policies, legal provisions, or branch-specific regulations often require access to plaintext emails for threat analysis, spam filtering, or mandatory business communication archiving [43]. For compliance reasons and efficient certificate management, such access is usually granted on a central instance, precluding client end-to-end encryption. So-called *Secure Email Gateways* allow the upload of key material to encrypt and decrypt emails before they transfer them to external contacts or the mailbox owner. Consequently, this service has full access to the decrypted message and may be vulnerable to decryption oracle attacks.

For the sender, such a gateway acts as a regular SMTP server. The Simple Mail Transfer Protocol (SMTP) [28], in combination with commons extensions introduced by the Extended Simple Mail Transfer Protocol (ESMTP), is the primary way to transmit emails. It follows a line-based command/reply model. A client uses successive commands to submit the email to the server, who answers with specific reply codes, indicating if the command was successful or not.

The email gateway can either use SMTP replies during transmission or return — *bounce* — the email afterward to inform the sender about an error. A successful oracle attack requires one of these mechanisms to provide enough information about the format of the decrypted message to the attacker.

We found a practical attack against Google's *Hosted S/MIME* functionality that shows the practicability of format oracle attacks in email E2EE in specific scenarios.

A.1 Hosted S/MIME

Google offers *Hosted S/MIME*[4] functionality for Google Workspace. Users can upload private keys to view S/MIME-encrypted emails in plaintext in the Gmail web interface or retrieve them via IMAP. In August 2020, Google implemented the SMTP error response listed in Listing 7 on their SMTP servers to reject encrypted messages without a valid signature.

This mechanism prevents malicious ciphertext modifications, typically used for Efail exploits [35]. However, such behavior forms an oracle that signals if a given S/MIME encrypted email contains a valid inner signature.

[4] "Enable hosted S/MIME for message encryption" https://support.google.com/a/answer/6374496 (accessed 2022-05-31).

A.2 CBC Padding Oracle Attack

The service decrypts the ciphertext to extract and validate the inner signature. Most S/MIME messages use the CBC encryption mode with PKCS #5 padding. Therefore, for successful decryption, this padding has to be valid.

We can use this behavior to create a CBC padding oracle [44] $O(c)$ that allows us to decrypt any S/MIME message encrypted for a Google Hosted S/MIME account. First, the attacker invalidates the inner signature of the encrypted email. As inner signatures are usually within the last blocks of an S/MIME ciphertext, an attacker can invalidate them by removing the final block, which truncates the signature, and forces the signature verification to fail. As a result, the email contains either a ciphertext with no inner signature or an invalid inner signature. The Gmail SMTP server will reject both upon successful decryption.

For the actual attack, the attacker increments the last byte of the penultimate block of an S/MIME ciphertext and sends the tampered ciphertext to the Gmail SMTP Server. The server immediately decrypts the message. If the PKCS #5 padding within the modified plaintext is *invalid*, the Gmail service will *accept* the message. If the padding is *valid*, the Gmail service tries to validate the inner signature, which fails, and *rejects* the message with the SMTP error code *554-5.7.5*.

$$O(c) = \begin{cases} \text{decryption failed} & \text{mail accepted} \\ \text{decryption succeeded} & \text{SMTP error } \textit{554-5.7.5} \end{cases}$$

The attacker repeats this process for all possible byte values or until they get the SMTP error code *554-5.7.5* (on average, 128 trials per plaintext byte)[5]. The attacker has now learned the last byte of the message and continues the process to decrypt the other bytes.

After our report, Google resolved the issue by no longer-bouncing unsigned messages. Instead, they now mark unsigned emails as suspicious, which disables automatic image loads and serves as a stopgap measure against Efail.

This vulnerability proves that format oracle attacks against S/MIME encrypted emails are realistic under certain conditions. In this case, it was a seemingly unrelated change that severely impedes the confidentiality of encrypted messages.

A.3 Countermeasures

Section 6.3 in the RFC5321 [28] (SMTP) discusses several ways of dealing with unsolicited and hostile messages. In the context of this paper, it is evident that any oracle behavior should be prevented. On the other hand, silent message dropping without informing the sender should only be considered in rare cases. A good way of handling encryption and format errors might be to notify the receiver about blocking potential fraudulent messages.

[5] Google rate limits IP addresses after about 1,000 requests. This can be circumvented by changing IP addresses after roughly 8 decrypted bytes.

B Details on Client Selection and Evaluation of External Content Loading

B.1 Client Selection

Since we restricted our analysis to S/MIME capable MUAs, we selected clients that support S/MIME based on prior work in [35]. We excluded long outdated clients, namely:

- Outlook 2007 to 2013
- Windows 10 Mail
- Windows Live Mail
- IBM Notes

We list the remaining clients and tested versions in Table 3.

B.2 External Content Loading

In addition to evaluating email clients' handling of multiple encrypted parts, fetching behavior, and decryption time, we analyzed all clients for their support of external content. We report whether clients show external content in emails at all and if they display it upon opening the email or if they require explicit user interaction, e.g., acknowledging a privacy warning. We report the results in the third column of Table 3.

Only The Bat!, Airmail, and macOS Mail load external content in emails by default. While some clients enable external content for specified senders, we could not distinguish between correct and incorrect formats in encrypted emails through timing measurements of external content requests. Therefore, they did not help our format oracle attacks.

We assume that most clients block automatic external content loading due to privacy concerns. Notably, our observations differ from those of Poddebniak et al. in 2018 [35]. We assume that some developers have taken steps to mitigate these attacks or became aware of privacy implications only after the publication.

Client	Version	External Content Support
Windows		
eM Client	8.2.1473	○
Outlook 2016	2108	○
Outlook 2019	2108	○
Postbox	7.0.49	○
The Bat!	9.4.4	◑
Cross-Platform (tested on Linux)		
Claws	4.0.0	–
Mutt	2.1.3	–
Thunderbird	91.1.2	○
Trojitá	0.7-5	○
Linux		
Evolution	3.40.4	○
KMail	5.18.1	○
macOS		
Airmail	5.0.7	◑
Mail	macOS 11.6	◑
MailMate	1.13.2	○
iOS		
Mail	iOS 15.6	○
Android		
MailDroid	5.09	○
Nine	4.9.1b	○
R2Mail2	2.54.305	○
Web		
Horde IMP	6.2.27	○

Legend	
–	Not supported.
○	Needs explicit user interaction to load.
◑	Loads upon opening email.
●	Loads automatically in the background.

Table 3: **Results of our External Content Evaluation.** We report if clients support external content and when it is loaded.

C List of Relevant Code Sections in Cryptographic Libraries

C.1 GnuPG

- PKCS#1v1.5 padding checks for OpenPGP messages: https://github.com/gpg/gnupg/blob/25ae80b8eb6e9011049d76440ad7d250c1d02f7c/g10/pubkey-enc.c, lines 280 to 377.
- PKCS#1 v1.5 padding checks for S/MIME messages: https://github.com/gpg/gnupg/blob/25ae80b8eb6e9011049d76440ad7d250c1d02f7c/sm/decrypt.c, lines 855 to 883.

C.2 OpenPGP.js

- PKCS#1 v1.5 padding checks: https://github.com/openpgpjs/openpgpjs/blob/39aa742c7ab5a61f07bcf30fb7e3daa34ae8ad8e/src/crypto/pkcs1.js, lines 97 to 114.
- Check of the encoded data inside a public key encrypted session key packet: https://github.com/openpgpjs/openpgpjs/blob/31fe960261519944b00a0d9d9887abd3ef863c22/src/packet/public_key_encrypted_session_key.js, lines 112 to 136.

C.3 Mozilla NSS

- PKCS#1 v1.5 padding checks: https://github.com/nss-dev/nss/blob/9bb9f91dc8a41852122e623d66cf5217b239b42a/lib/freebl/rsapkcs.c, lines 1091 to 1215.
- Key validation function of Mozilla NSS: https://github.com/nss-dev/nss/blob/9bb9f91dc8a41852122e623d66cf5217b239b42a/lib/softoken/pkcs11.c, lines 1310 to 1425.
- CBC padding checks for S/MIME messages: https://github.com/nss-dev/nss/blob/9dab43371d4d924419523e18ba84f02804880533/lib/smime/cmscipher.c, lines 365 to 540.